.UBAfiuba
FACULTAD DE INGENIERÍA

Teoría de Algoritmos
(75.29) Curso Buchwald - Genender

Trabajo Práctico 2
Mateo empieza a Jugar

2 de diciembre de 2024

Damaris Juares
108566

Lucas Perez Esnaola
107990

Índice

1. Introducción

1.1. Contexto

Pasan los años. Mateo ahora tiene 7 años. Los mismos años que tenía Sophia cuando comenzaron a jugar al juego de las monedas. Eso quiere decir que Mateo también ya aprendió sobre algoritmos greedy, y lo comenzó a aplicar. Esto hace que ahora quién gane dependa más de quién comience y un tanto de suerte.

Esto no le gusta nada a Sophia. Ella quiere estar segura de ganar siempre. Lo bueno es que ella comenzó a aprender sobre programación dinámica. Ahora va a aplicar esta nueva técnica para asegurarse ganar siempre que pueda

1.2. Consigna

1. Hacer un análisis del problema, plantear la ecuación de recurrencia correspondiente y proponer un algoritmo por programación dinámica que obtenga la solución óptima al problema planteado: Dada la secuencia de monedas $m_1, m_2, ..., m_n$, sabiendo que Sophia empieza el juego y que Mateo siempre elegirá la moneda más grande para sí entre la primera y la última moneda en sus respectivos turnos, definir qué monedas debe elegir Sophia para asegurarse obtener el máximo valor acumulado posible. Esto no necesariamente le asegurará a Sophia ganar, ya que puede ser que esto no sea obtenible, dado por cómo juega Mateo. Por ejemplo, para [1, 10, 5], no importa lo que haga Sophia, Mateo ganará.

2. Demostrar que la ecuación de recurrencia planteada en el punto anterior en efecto nos lleva a obtener el máximo valor acumulado posible.

3. Escribir el algoritmo planteado. Describir y justificar la complejidad de dicho algoritmo. Analizar si (y cómo) afecta a los tiempos del algoritmo planteado la variabilidad de los valores de las monedas.

4. Realizar ejemplos de ejecución para encontrar soluciones y corroborar lo encontrado. Adicionalmente, el curso proveerá con algunos casos particulares que deben cumplirse su optimalidad también.

5. Hacer mediciones de tiempos para corroborar la complejidad teórica indicada. Agregar los casos de prueba necesarios para dicha corroboración (generando sus propios sets de datos). Esta corroboración empírica debe realizarse confeccionando gráficos correspondientes, y utilizando la técnica de cuadrados mínimos. Para esto, proveemos una explicación detallada, en conjunto de ejemplos

6. Agregar cualquier conclusión que parezca relevante.

2. Análisis del Problema

2.1. Descripción de problema

Sofía y Mateo están jugando a un juego de monedas. La secuencia de monedas es $m_1, m_2, \ldots, m_n$. En cada turno, Sofía o Mateo eligen una moneda de una fila de monedas que está a la izquierda o a la derecha. El objetivo de Sofía es maximizar su valor acumulado. Sin embargo, Mateo, que siempre juega con la estrategia más agresiva, siempre tomará la moneda más grande de las dos opciones disponibles (de la izquierda o de la derecha) en su turno.

Sofía comienza primero. Aunque Sofía intente maximizar su valor, la estrategia de Mateo puede hacer que, en algunos casos, ella no pueda ganar. Por ejemplo, en una secuencia de monedas como $[1, 10, 5]$, no importa lo que haga Sofía, Mateo ganará debido a la forma en que elige las monedas. Este planteo busca que por cada moneda que tenga la opcion de elegit, poder analizar como impactara eso en toda la jugada global, es decir, resolviendo los subproblemas que se desprenden de cada posible decisión.

2.2. Propuesta de algoritmo dinamico

Nuestro objetivo es poder adelantarnos a la posible futura jugada de Mateo, ya que, siempre elegira la mejor opcion localmente, pero Sofia puede adelantarse usando programación dinamica y ver que monedas son mas beneficiosas y asegurarse de no dejarselas disponibles para Mateo.

Ejemplo: Si tuvieran las siguientes monedas n= [70, 80, 27, 30]

Sofia en este caso tiene que poder adelantarse estrategicamente hablando y hacer el correcto analisis de las que estan disponible en su turno y las que estaran disponibles en los siguientes.

En este caso, como Sofia es mas grande y mas inteligente usara el razonamiento para darse cuenta que si elige la moneda[0]= 70 en lugar de moneda[-1]=30 eso le traera un problema a futuro cuando Mateo en su turno aplique su estrategia y se lleve la moneda mas grande de toda la partida (80).

Con eleccion codiciosa el resultado seria:

Monedas Sofia= 70, 30. Suma= 100

Monedas Mateo= 80, 27. Suma= 107

Con eleccion dinamica el resultado se veria asi:

Monedas Sofia= 30, 80. Suma= 110

Monedas Mateo= 70, 27. Suma= 97

Es decir, Sofia se adelanto a la siguiente jugada.

La estrategia es no tomar una decision como en la partida anterior (version codiciosa) solo teniendo en cuenta las 2 monedas actuales disponibles sino que pensar en las posibles monedas que estaran disponibles en los siguientes turnos.

2.2.1. Ecuación de Recurrencia

Definimos $dp[i][j]$ como el valor máximo que Sofía puede obtener si juega con las monedas en el rango de índices $[i, j]$, es decir, si tiene que tomar una decisión sobre qué monedas seleccionar entre m_i y m_j.

2.2.2. Caso base

Cuando solo queda una moneda en el rango, el valor máximo es simplemente el valor de esa moneda:

$$dp[i][i] = m_i$$

2.2.3. Caso recursivo

Cuando hay más de una moneda en el rango, Sofía tiene dos opciones: elegir la primera moneda m_i o la última m_j. Sin embargo, debe tener en cuenta que Mateo, en su turno, siempre tomará la moneda más grande entre las dos opciones disponibles (de la izquierda o de la derecha).

- Si Sofía elige m_i, el siguiente turno es de Mateo. Mateo entonces elegirá entre m_{i+1} y m_j, y el valor total que Sofía podrá obtener será el valor de m_i más el valor restante de las monedas después de que Mateo elija. - Si Sofía elige m_j, el siguiente turno es de Mateo, quien elegirá entre m_i y m_{j-1}, y el valor total que Sofía podrá obtener será el valor de m_j más el valor restante después de que Mateo elija.

Entonces, la ecuación de recurrencia es:

$$dp[i][j] = \text{máx}\left(m_i + \text{máx}(dp[i+2][j], dp[i+1][j-1]), m_j + \text{máx}(dp[i+1][j-1], dp[i][j-2])\right)$$

La idea detrás de la recurrencia es que Sofía busca maximizar su valor actual, pero debe anticipar las elecciones de Mateo, por lo que la estrategia es elegir la opción que deje a Mateo con las peores opciones posibles (es decir, la que minimice lo que Mateo puede obtener después de su turno).

3. Demostración de la Ecuación de Recurrencia

Para demostrar que la ecuación de recurrencia planteada nos lleva a obtener el máximo valor acumulado posible, debemos realizar un análisis basado en la estructura de las decisiones de Sofía y Mateo, y cómo estas afectan a la función $dp[i][j]$.

3.1. Planteamiento de la Ecuación de Recurrencia

La ecuación de recurrencia planteada es la siguiente:

$$dp[i][j] = \text{máx}\left(m_i + \text{máx}(dp[i+2][j], dp[i+1][j-1]), m_j + \text{máx}(dp[i+1][j-1], dp[i][j-2])\right)$$

Donde:

- $dp[i][j]$ es el valor máximo que Sofía puede obtener al jugar con las monedas en el rango de índices $[i, j]$.
- m_i y m_j son las monedas en las posiciones i y j, respectivamente.
- La función $\text{máx}(dp[i+2][j], dp[i+1][j-1])$ representa tambien el valor que puede obtener Sofía si elige m_i, ya que después de su elección, Mateo puede elegir entre las opciones $[i+2, j]$ o $[i+1, j-1]$ para minimizar la ganancia de Sofía.
- La función $\text{máx}(dp[i+1][j-1], dp[i][j-2])$ representa el valor que puede obtener Sofía si elige m_j, ya que Mateo puede elegir entre las opciones $[i+1, j-1]$ o $[i, j-2]$ para minimizar la ganancia de Sofía.

3.2. Demostración por Inducción

Para demostrar que esta ecuación lleva efectivamente al valor óptimo, utilizaremos inducción sobre el tamaño del subproblema, es decir, sobre la longitud del rango de monedas considerado.

3.2.1. Caso base

Cuando hay solo una moneda en el rango $[i, i]$, es claro que Sofía tomará esa moneda, ya que no hay otra opción. En este caso, tenemos:

$$dp[i][i] = m_i$$

Este es el caso base, que está de acuerdo con la ecuación de recurrencia planteada, ya que no hay decisiones posteriores para tomar.

3.2.2. Paso inductivo

Supongamos que la ecuación de recurrencia es válida para rangos de monedas más pequeños. Ahora, consideremos un rango de longitud mayor, $[i, j]$, con $j > i$.

Sofía tiene dos opciones:

1. **Elegir la moneda m_i:** En este caso, Mateo elegirá entre dos rangos: $[i+2, j]$ o $[i+1, j-1]$. La elección de Mateo afectará el valor total de las monedas restantes. La ecuación de recurrencia toma el valor máximo entre estas dos opciones para asegurar que Sofía obtenga el mejor resultado posible en este escenario. Luego, Sofía suma el valor de m_i a la ganancia que Mateo podría dejarle, lo cual se expresa como:

$$m_i + \text{máx}(dp[i+2][j], dp[i+1][j-1])$$

2. **Elegir la moneda m_j:** En este caso, Mateo elegirá entre los rangos $[i+1, j-1]$ o $[i, j-2]$. De manera similar, la ecuación de recurrencia calcula la ganancia para Sofía después de la elección de Mateo, y Sofía suma el valor de m_j. Esto se expresa como:

$$m_j + \text{máx}(dp[i+1][j-1], dp[i][j-2])$$

Finalmente, Sofía seleccionará la opción que maximice su ganancia:

$$dp[i][j] = \text{máx}\left(m_i + \text{máx}(dp[i+2][j], dp[i+1][j-1]), m_j + \text{máx}(dp[i+1][j-1], dp[i][j-2])\right)$$

3.3. Correctitud de la Ecuación

La ecuación de recurrencia tiene en cuenta las decisiones de Sofía y Mateo de manera adecuada. Sofía busca maximizar su valor total, pero debe anticipar las elecciones de Mateo, quien siempre elegirá la opción que minimice la ganancia de Sofía. Al elegir la moneda más grande disponible, Mateo sigue una estrategia agresiva que minimiza la opción de Sofía. La ecuación asegura que Sofía tome la mejor decisión posible en cada caso, maximizando su ganancia acumulada.

3.4. Conclusión

La ecuación de recurrencia planteada es correcta porque sigue la lógica del juego y permite calcular el valor máximo que Sofía puede obtener. Al considerar las decisiones de ambos jugadores, y al calcular el valor máximo para cada subproblema de manera recursiva, garantizamos que se obtiene el valor óptimo para Sofía al final del juego. La solución se construye correctamente utilizando la programación dinámica y la ecuación de recurrencia permite obtener la máxima ganancia acumulada posible para Sofía en cualquier secuencia de monedas.

4. Algoritmo en Programación Dinámica

El algoritmo sigue la idea de llenar una tabla dp donde cada celda $dp[i][j]$ almacena el valor máximo que Sofía puede obtener al jugar con las monedas entre m_i y m_j. La solución óptima será el valor almacenado en $dp[0][n-1]$, donde n es el número total de monedas.

4.1. Código

```
def pd(monedas):
    n = len(monedas)
    # Crear tabla bidimensional para almacenar los resultados
    dp = [[0] * n for _ in range(n)]

    # Llenar la tabla base: casos con una sola moneda
    for i in range(n):
        dp[i][i] = monedas[i]

    # Llenar la tabla para rangos crecientes de tama o
    for longitud in range(2, n + 1):    # Tama o del subarreglo
        for inicio in range(n - longitud + 1):  # Inicio del rango
            fin = inicio + longitud - 1         # Fin del rango

            # Calcular la mejor elecci n para el rango [inicio, fin]
            elegir_inicio = monedas[inicio] + max(
                dp[inicio + 2][fin] if (inicio + 2 <= fin) and (monedas[inicio+1]>
    monedas[fin]) else 0,
                dp[inicio + 1][fin - 1] if (inicio ++ 1 <= fin - 1) and (monedas[
    inicio+1]<monedas[fin])else 0,
            )
            elegir_fin = monedas[fin] + max(
                dp[inicio][fin - 2] if (inicio <= fin - 2) and (monedas[inicio]<
    monedas[fin-1])  else 0,
                dp[inicio + 1][fin - 1] if (inicio + 1) <= fin - 1 and (monedas[
    inicio]>monedas[fin-1]) else 0,
            )
            dp[inicio][fin] = max(elegir_inicio,  elegir_fin )

    # El resultado est  en dp[0][n-1], considerando todas las monedas

    print(f"Ganancia Sofia: {dp[0][n-1]} \nGanancia Mateo: {sum(monedas)-dp[0][n
    -1]}")
```

4.2. Análisis de la complejidad

- Llenamos la tabla para rangos crecientes de tamaño en $O(n^2)$.
- Calculamos la mejor elección para el rango [inicio, fin] usando la tabla, accediendo a ella en $O(1)$.
- Comparamos las decisiones en $O(1)$.
- Se guarda la mejor opción y la decisión tomada en la tabla en $0(1)$.
- La tabla dp es una matriz bidimensional de tamaño nxn lo que requiere $O(n^2)$ de espacio.

Por lo tanto, la complejidad temporal total es $O(n^2)$ y la complejidad espacial es $O(n^2)$.

4.3. Análisis del impacto de la variabilidad de los valores de las monedas en los tiempos del algoritmo

El tiempo de ejecución del algoritmo **no depende directamente de los valores de las monedas**, sino de la cantidad de monedas (n) debido a las siguientes razones:

4.3.1. Acceso constante a las tablas `dp` y `decisiones`

El algoritmo llena la tabla `dp` realizando operaciones en tiempo constante $O(1)$ para cada combinación de *inicio* y fin. Las comparaciones y selecciones de los valores máximos dependen de los índices y no del tamaño o magnitud de los valores numéricos de las monedas.

4.3.2. Impacto de las decisiones de Mateo

Aunque Mateo selecciona la moneda más grande entre las disponibles (primera o última), esta decisión implica una comparación en $O(1)$. La variabilidad en los valores de las monedas afecta **qué camino toma el algoritmo**, pero no aumenta la cantidad de operaciones realizadas. Independientemente de los valores, el número de iteraciones en los bucles es fijo y depende solo de n.

4.3.3. Relevancia de la distribución de valores

Si los valores de las monedas están distribuidos de forma uniforme o si hay grandes disparidades, las elecciones de Mateo y Sophia podrían ser más evidentes o requerir más análisis manual, pero **esto no afecta la complejidad computacional**. El algoritmo siempre realiza las mismas comparaciones en cada rango definido por *inicio* y fin, independientemente de la magnitud o distribución de los valores.

4.3.4. Conclusión

La variabilidad de los valores de las monedas **no afecta los tiempos del algoritmo en términos de complejidad asintótica**, ya que todas las operaciones involucradas (comparaciones y actualizaciones de tablas) son independientes de los valores específicos.

Sin embargo, en aplicaciones prácticas, una mayor variabilidad en los valores podría influir en el resultado de las elecciones y en la percepción de eficiencia, pero no en la cantidad de operaciones realizadas.

Por lo tanto, la complejidad temporal y espacial del algoritmo sigue siendo $O(n^2)$, sin importar cómo varíen los valores de las monedas.

5. Análisis temporal

Para corroborar la complejidad teórica indicada ($O(n^2)$), se medirán los tiempos de ejecución del algoritmo de programación dinámica planteado con sets de datos generados aleatoriamente con distintos tamaños.

5.1. Generador de sets aleatorios

Para generar los sets aleatorios, se utilizó la siguiente función:

```python
# Genera n sets de datos de tamano incremental para el problema de las monedas.
# Si se especifica una semilla, se utilizara para generar los datos.
def generar_set_datos_monedas(semilla, n):
    if semilla != None:
        seed(semilla)

    set_datos = []

    for i in range(1, n+1):
        set_datos.append([randint(1, 10000) for _ in range(i * 10)])

    return set_datos
```

De esta forma, se generan n sets de datos que van incrementando su tamaño iterativamente.

5.2. Mediciones

Generamos 100 sets de datos, donde cada set incrementa en 10 su tamaño frente al anterior. Las mediciones temporales al ejecutar el algoritmo de programación dinámica se muestran a continuación:

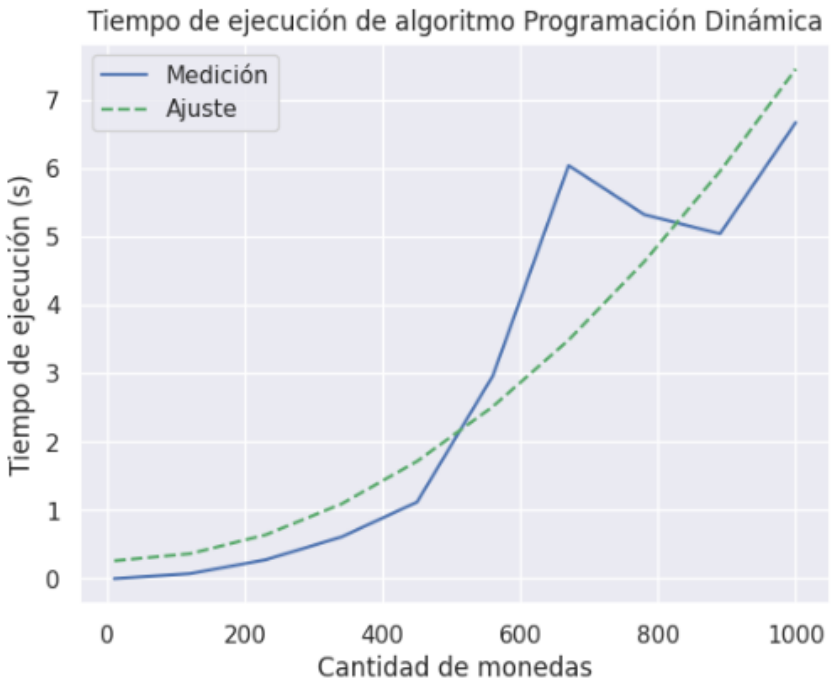

Figura 1: Mediciones temporales para el algoritmo planteado

Para corroborar la complejidad teórica, se utilizó la técnica de cuadrados mínimos para ajustar los datos medidos a una recta cuadrática. Podemos ver como la recta de ajuste cuadrático ajusta en terminos generales de manera adecuada a las mediciones, por lo que parecería confirmar que la complejidad teórica calculada es correcta.

Para visualizar mejor el resultado del ajuste, podemos graficar el error absoluto del ajuste segun el tamaño de entrada.

Figura 2: Error absoluto del ajuste según el tamaño de entrada

En este caso, notamos que el error absoluto para el ajuste si fue de un valor considerable. Podemos ver como en términos generales el error absoluta ronda el valor de 1.0, sin embargo para ciertos tamaños de entradas tiene un pico en 2.5. Esto coincide con el gráfico de mediciones ya que en el mismo intervalo el ajuste se desvía más de la medición. Entendemos que estos son casos particulares y que, en general, el ajuste fue bueno y por lo tanto la complejidad teórica indicada es correcta.

6. Conclusiones

El algoritmo planteado utiliza programación dinámica para asegurar que Sofía maximice su valor acumulado, anticipando las decisiones de Mateo. La complejidad temporal del algoritmo es $O(n^2)$, y la espacial es también $O(n^2)$, debido a las tablas *dp* y *decisiones*. La estrategia optimiza las elecciones de Sofía, aunque en algunos casos puede que ella no gane debido a la forma en que Mateo elige sus monedas.